version 0.6.20120126

- a debugger for StarSs

Steffen Brinkmann
HLRS, Universität Stuttgart

Disclaimer: The information contained in this manual is not garanteed to be complete at this stage. It is subject to changes without further notice. Please send comments, corrections and additional text to brinkmann@hlrs.de.

Steffen Brinkmann: *TEMANEJO - a debugger for StarSs*, manual for the graphical debugging tool TEMANEJO .

Revision history:

Version	Date	Remarks	Author
1.0	6 Sep 2012	initial version	Steffen Brinkmann

© 2009-2012, HLRS, University of Stuttgart, all rights reserved.

Redistribution and use in source (SGML DocBook) and 'compiled' forms (SGML, HTML, PDF, PostScript, RTF and so forth) with or without modification, are permitted provided that the following conditions are met:

Redistributions of source code (SGML DocBook) must retain the above copyright notice, this list of conditions and the following disclaimer as the first lines of this file unmodified.

Redistributions in compiled form (transformed to other DTDs, converted to PDF, PostScript, RTF and other formats) must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.

THIS DOCUMENTATION IS PROVIDED BY THE COPYRIGHT HOLDER "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS DOCUMENTATION, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

Contents

1 Introduction **5**

2 Quick start guide for SMPSs **5**

3 Installing TEMANEJO **6**
- 3.1 Linux . . . 6
  - 3.1.1 Ubuntu . . . 7
  - 3.1.2 Gentoo . . . 7
- 3.2 Mac OS . . . 7

4 Hands on - the users guide **8**
- 4.1 Starting TEMANEJO . . . 8
  - 4.1.1 Local session . . . 8
  - 4.1.2 Remote session . . . 10
- 4.2 The graphical user interface . . . 10
  - 4.2.1 The graph display . . . 11
  - 4.2.2 The information and control display . . . 12
  - 4.2.3 The toolbar . . . 13
  - 4.2.4 The status bar . . . 14
- 4.3 How to invoke the gnu debugger . . . 14

5 Behind the scenes - the technical guide **15**
- 5.1 General information . . . 15
- 5.2 Setting up the socket connection . . . 15
- 5.3 The information stream . . . 16
  - 5.3.1 The format . . . 16
  - 5.3.2 The events . . . 17

1 Introduction

How to read this manual

TEMANEJO is designed to be widely self-explanatory. Therefore you might want to read the *Quick start guide* (section 2) and start debugging your SMPSs application right away.

If you have trouble installing TEMANEJO on your specific architecture section 3 about *Installing TEMANEJO* might be the point to start for you.

For more in depth information of how to get most out of TEMANEJO read *Hands on* (section 4).

The section *Behind the scenes* (section 5) is the one you would want to read if you plan to instrument a StarSs runtime, expand TEMANEJO or alike.

2 Quick start guide for SMPSs

If you want to get started as quickly as possible follow these few steps:

1. Make sure that the python module NetworkX is installed on the computer where TEMANEJO is to be run (http://networkx.lanl.gov/).
2. Make sure your SMPSs distribution is instrumented with calls to `AYU_EVENT`. To check simply type

   shell
   ```
   # grep AYU /path/to/SMPSs/master/*
   ```

   the output should by about 20 occurrences of `AYU_EVENT`.
3. Go to the SMPSs directory and type (if configure is already present in the directory you can skip the first line):

   shell: Ayudame directory
   ```
   # autoreconf -fiv
   # ./configure --with-flavour=smp --with-ayudame=<AYUDAME_DIR>
   ```

   `AYUDAME_DIR` is the path to the Ayudame directory. Make sure not to use ~ and alike for specifying the directory but rather the absolute path.
4. Go to the `AYUDAME_DIR` directory (usually /path/to/SMPSs/../../Tools/Ayudame) and compile and install `libayudame.so`

   shell: Ayudame directory
   ```
   # make
   # make install
   ```

   This will create `libayudame.so` and install it and the AYUDAME headers in PREFIX/lib and PREFIX/include respectively. (default for PREFIX is the current directory)

5. compile SMPSs with Ayudame support. In the SMPSs directory, type:

```
shell: SMPSs directory
# make && make install
```

6. compile your program

```
shell: application directory
# smpss-cc smpss_app.c -o smpss_app
```

7. Run TEMANEJO:

```
shell: Ayudame directory
# ./temanejo.py /path/to/smpss_app
```

and press Connect. The dialogue disappears and you will want to press "Run" (the triangular shaped "Play"-button in the toolbar).

3 Installing TEMANEJO

TEMANEJO requires python version >= 2.4 and some additional packages which are freely downloadable for most common systems. In case your system is not mentioned below or you want to get newer versions than the ones that come with your system, please visit the respective websites:

Python http://www.python.org/ - recommended version 2.7.2

PyGTK http://www.pygtk.org/ - recommended version 2.24.0

PyGObject http://live.gnome.org/PyGObject - recommended version 3.0.0

PyCairo http://cairographics.org/pycairo/ - recommended version 1.8.8

NetworkX http://networkx.lanl.gov/ - recommended version 1.5

Please note that these software packages may depend on other software such as C libraries etc. In the following you find the package names for different Linux[1] distributions and a short "Howto" for Mac OS[2].

3.1 Linux

TEMANEJO is easy to install on Linux systems. Usually some extra packages need to be installed. You find the names for these packages for some distributions below.

[1] Linux is a trademark owned by Linus Torvalds
[2] Mac OS is a trademark of Apple Inc.

3.1.1 Ubuntu

Install the following packages with

shell: as superuser

```
# apt-get install <package>
```

- **python**
- **python-gtk2**
- **python-gobject**
- **python-cairo**
- **python-networkx**

3.1.2 Gentoo

Install the following packages with

shell: as superuser

```
# emerge <package>
```

- **dev-lang/python**
- **dev-python/pygtk**
- **dev-python/pygobject**
- **dev-python/pycairo**
- **dev-python/networkx**

3.2 Mac OS

Most package managers, as MacPorts and Fink, will provide packages for these requirements. The rest of this guide assumes you are using MacPorts (http://www.macports.org). The same basic procedure applies for other package managers or source installation.

To install all required packages safely, you will make sure your MacPorts is up-to-date, install Python v2.7 (this will take a while) and install the requirements for TEMANEJO :

shell: as superuser

```
# port selfupdate
# port install python27
# port install py27-gtk py27-networkx
```

4 Hands on - the users guide

4.1 Starting TEMANEJO

There are two ways to start your TEMANEJO session. The difference lies in how you connect TEMANEJO to you application. The first way is to let TEMANEJO start your application on the same machine that TEMANEJO is running on. Typically you will use this way for running the application and debugging it on you laptop or desktop machine. We will refer to this debugging mode as *"local"*.

The other way is to start your application and TEMANEJO independently and specify the hostname and port via which TEMANEJO will connect to the application. In this case, the application can run on a remote computer (or computing node) as long as is accessible from your local system via network. This type of connection will be referred to as *"remote"*. Please note that the remote machine may as well be "localhost".

4.1.1 Local session

For a local debugging session simply invoke temanejo.py from the command line:

shell
```
# temanejo.py
```

You may as well place TEMANEJO 's icon in your launch bar or on you desktop and start it with a simple click.

Figure 1: TEMANEJO icon

TEMANEJO will open the connect dialogue automatically. In this dialogue you must specify the application you want to run. TEMANEJO also needs to know the location of the AYUDAME library. When the installation process didn't fail, this library is found at the default place and you do not have to change this parameter.

All other settings are optional. The optional settings are: the command line options for your application, the configuration file for SMPSs, the port and the number of threads you want to run your application on.

For convenience, you can specify the application which you want to debug in the command line, appending it to the call to TEMANEJO :

shell
```
# temanejo.py my_smpss_app
```

There are many command line options to TEMANEJO which will find in the help string:

shell
```
# temanejo.py --help
Usage: temanejo.py [options] [smpss_app [smpss_app_args]]
```

```
TEMANEJO is designed to connect to an SMPSs instrumented application which has
been linked to the AYUDAME library. The SMPSs application can either be run
independently on the same or a remote computer, or can be given to TEMANEJO as
a command line option and started by TEMANEJO. Please refer to the README file
and the documentation in the doc/ directory.

Options:
  --version             show program's version number and exit
  -h, --help            show this help message and exit
  -v, --verbose         Switch on verbose output. Use -vv and -vvv for
                        increasingly epic information
  -q, --quiet           Switch off any output except error messages
  -t TITLE, --title=TITLE
                        window title (appended to "TEMANEJO:")
  -d LAYOUTDIR, --layout-direction=LAYOUTDIR
                        how to draw the graph: top to bottom ('t2b'), left to
                        right ('l2r'),Bottom to top ('b2t') or right to left
                        ('r2l') [default: t2b]
  -Z, --autozoom        Switch on autozooming of the graph
  -z, --no-autozoom     Switch off autozooming of the graph [default]
  -n NODE_COL, --node-colour=NODE_COL
                        Choose what the node colour means. One of "nothing"
                        (no colours drawn), "status" (colour indicates task
                        status), "thread" (thread Id), "function" [default]
                        (function Id), "task_dur" (task duration)
  -m MARGIN_COL, --margin-colour=MARGIN_COL
                        Choose what the margin colour means. One of "nothing"
                        (no margins drawn), "status" [default] (margin
                        indicates task status), "thread" (thread Id),
                        "function" (function Id), "task_dur" (task duration)
  -e EDGE_COL, --edge-colour=EDGE_COL
                        Choose what the edge colour means. One of "nothing"
                        (black edges), "orig_addr" [default] (original address
                        of dependency), "addr" (address of dependency after
                        renaming)
  -s SHAPES, --shapes=SHAPES
                        Choose what the node shape means. One of "nothing" (no
                        margins drawn), "thread" [default] (thread Id),
                        "function" (function Id)
  -L LIB_FILE, --library=LIB_FILE
                        Specify library file to communicate with the
                        application. By default, libayudame.so is expected in
                        TEMANEJO's directory.
  --pygraphviz          Specify whether to use pygraphviz for the graph
                        layout. Pygraphviz is part of the networkx package,
                        but may cause problems on some systems.
  --global-tasks-in-first-row
                        Specify whether global tasks should be drawn in first
```

```
                        line.

TEMANEJO is published under the terms of the BSD license. (C) HLRS, University
of Stuttgart. For further information please refer to the LICENSE file.
```

4.1.2 Remote session

In order to start a remote session, you have to start your application with some environment variables set. Choose an unused port and set `AYU_PORT`:

shell

```
# export AYU_PORT=5977
```

Usually port numbers higher than 1000 are safe. To be sure type:

shell

```
# netstat -an|grep -i listen
```

and use a port which is not in the list. Then start your application preloading the AYUDAME library:

shell

```
# LD_PRELOAD=/path/to/libayudame.so ./my_smpss_app
```

The application will run until it reaches `#pragma css start`. Now, launch TEMANEJO .

shell

```
# temanejo.py
```

In the connection dialogue, choose the "remote" tab and set the host and port to the corresponding values. If you exported `AYU_PORT` before launching TEMANEJO , the specified port number will automatically be set.

Now press "Connect" and debug your program.

4.2 The graphical user interface

The graphical user interface (GUI) is divided into two main sections: the graph display to the right and the information display to the left. Additionally there is a toolbar at the top and a status bar at the bottom of the window (see figure 2).

To get the most out of TEMANEJO you will interact with all parts of the GUI. In the following we will describe the full functionality of all parts of the program.

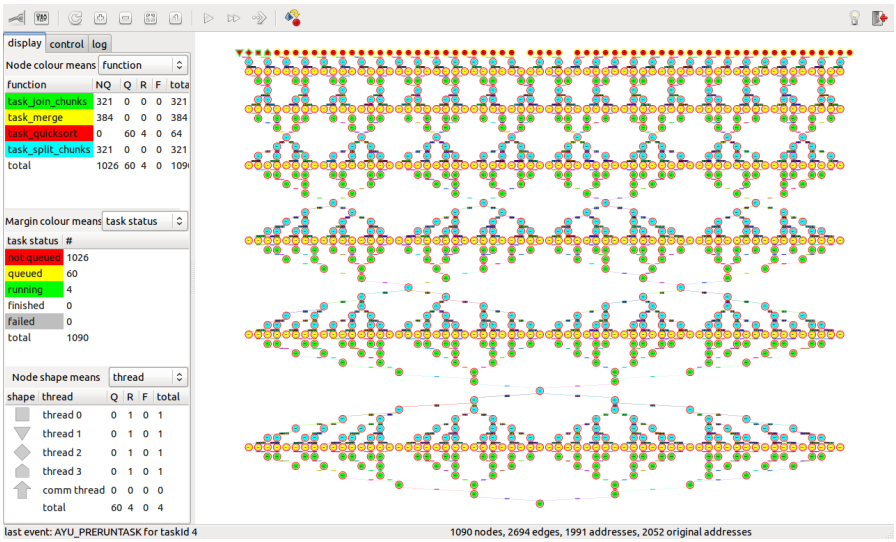

Figure 2: TEMANEJO screenshot

4.2.1 The graph display

The graph display, or *graph widget*, uses most part of the program surface. When starting TEMANEJO , it will be a white area. It will be filled with the dependency graph when the first task is added to it, i.e. when the first taskified function is reached in the program execution. Please note that after connecting to an application, TEMANEJO will pause execution of the application at the point of the `#pragma css start`. After you advance the application execution by at least one step (see section 4.2.3) the StarSs runtime will start adding tasks to the graph.

Looking at the whole graph: To change the view on the dependency graph you can use the following commands:

Action	Keyboard	Mouse
Zoom In	PageUp *or* +	Scroll wheel up
Zoom Out	PageDown *or* -	Scroll wheel down
Standard Zoom Level	z	
Magnify area		Hold middle button and mark rectangle
Move graph	*arrow buttons*	Hold left button and mark rectangle

Looking at a single task: With a right-click on a node (task), a context menu opens giving you more information about the task and offering some actions (see figure 3). The first two lines indicate the unique identification number of the task and the identification of the thread on which the task is or was executed or for which the task is scheduled. '-1' (as seen in figure 3) means that the task is not scheduled yet because there are still pending dependencies.

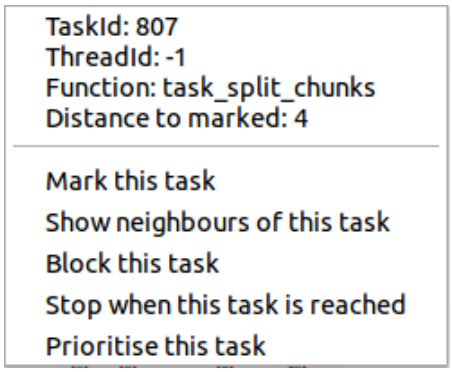

Figure 3: TEMANEJO context menu

The next line denotes the name of the function or a generated name of the form 'function N' where 'N' is a unique number. A function name is generated when the runtime environment did not send information about the actual function name (event `AYU_REGISTERTASK`, see section 5.3.2).

I a task is marked (see below), an additional line above the separator is displayed indicating the distance to the marked task. Distance means number of dependencies (edges) to or from the marked task. For tasks that do not depend on each other, i.e. could run in parallel potentially, the distance is denoted as 'None'. A negative number means, that the marked task is going to be executed before the clicked task.

The items below the separator are clickable and allow you to further analyse the task dependencies and to control the execution of the application. Please note that the latter depends on whether the StarSs framework supports these features. As of today, only SMPSs [**?**] does.

The first two clickable options of the context menu allow you to mark a task and to mark the task and highlight its direct neighbours, respectively. When you right-click on another task after having marked a task, the distance to that task is displayed in the context menu as described above.

The option 'Block this task' will mark the task with a thick red cross and send the request to block this task to the runtime. The intended behaviour of the StarSs runtime is to execute every task (and all code outside of tasks) which do not depend on the blocked task but not the blocked task itself.

The option 'Stop when this task is reached' can be used to advance the execution of the application until a certain task is about to be executed. The execution will then pause. Usually you will uncheck the 'Stop...' options in the control tab in the information and control display to the left (see section 4.2.2) after you marked a task to pause at. Then you will advance the execution using the step function (see section 4.2.3). When the marked task is reached, you will switch on one or more of the 'Stop...' options and step through the graph as usual.

4.2.2 The information and control display

The information and control display consists of several panels organised in tabs.

Display tab : In the default tab, named "display", the meaning of the node colour, margin colour and node shape are denoted and can be changed via the drop down menu at the top of each list.

In the default settings, the node colour means "function", i.e. the colour of each node refers to the function which will be executed. In the table, the function names of

the registered functions are displayed with the respective node colour as background. The counters indicate how many of each function calls (tasks) are "not queued" (`NQ`), "queued" (`Q`), "running" (actually "about to run", `R`) and "finished" (`F`). Last column and row show the totals for each function and status, respectively.

Changing the meaning of the colours and shapes is done by choosing a different value from the drop down menus. The columns and rows will change respectively and are self-explanatory.

Control tab : In this tab you find a number of ways to control the execution and display of the graph. Topmost you can navigate directly to a task of a specific number. With the next button you can clear any markers set via the context menu in the graph view.

In the section "Breakpoints" you can specify, when the execution of the program should be paused. A click on the "Run" button (see section 4.2.3) will run the program until of the events is reached, which you checked here.

Also you can run `gdb` with a breakpoint set to a specific function.

Log tab : In the log tab the log of the incoming messages is shown.

4.2.3 The toolbar

The symbols on the toolbar will change depending on your window manager, button style and operating system. From left to right, the buttons perform the following tasks:

- **Connect**: Open the Connect dialogue.
- **Preferences**: Open the preferences dialogue.
- **Redraw**: Redraw the graph.
- **Zoom in**: Increase zoom level.
- **Zoom out**: Decrease zoom level.
- **Fit graph to window**: Adjust zoom level, so that the whole graph fits into the window. Please be aware that for large graphs this may result in a blank screen, when there are more nodes to be drawn than your screen has pixels.
- **Default zoom level**: Set zoom level to 1.
- **Run until next pause**: Use this to step through the graph.
- **Run until nth pause**: This is used for fast-stepping through the graph. `n` can be adjusted in the preferences dialogue.
- **Debug**: start `gdb`.

- ▷ **Export image or animation**: Export an image or animation of the current graph.
- ▷ **About TEMANEJO** : Some information about TEMANEJO .
- ▷ **Quit**: Quit TEMANEJO .

4.2.4 The status bar

The status bar provides you with some additional information about the running application and the task graph. On the left the last registered incoming message is displayed. In the middle, some graph characteristics are listed. In detail that is the number of nodes, the number of nodes (i.e. tasks), edges (i.e. dependencies), dependency addresses and dependency addresses before renaming. Renaming is a feature of the SMPSs runtime. For further information please refer to the SMPSs documentation.

4.3 How to invoke the gnu debugger

In order to be able to attach `gdb` to a running process, the file `/etc/sysctl.d/10-ptrace.conf` (in LINUX) needs to be edited as root. There is only one parameter in the file which has to be set to 0. This is only necessary once and requires a restart of the computer to take action.

To use `gdb` with TEMANEJO , you have to compile your application with the options `-g` and `-k`:

shell: application directory

```
# smpss-cc -g -k smpss_app.c -o smpss_app
```

These options are needed to compile with debugging symbols and to keep the temporary files that SMPSs generates. Actually it is these files that you will be debugging.

Start the application and TEMANEJO as usual and press the debug button or 'd' on your keyboard. A window with the gdb session opens. Type in your breakpoints (for example "b func_name") or other `gdb` commands. Then type 'c' (or 'continue'). `gdb` returns the control to TEMANEJO . Now you can step through the task graph with temanejo, but keep an eye on the `gdb` window. When a breakpoint is reached, TEMANEJO will stop reacting. Nevertheless, when you hit the "run" button 20 times, it will apply these 20 steps later. So don't.

Step through lines of code with gdb (command: 's' or 'step' or whatever you find in `gdb`'s manual). When you're done with `gdb`, type 'c' in the `gdb` window again, and the control goes back to TEMANEJO .

5 Behind the scenes - the technical guide

5.1 General information

TEMANEJO is written in the `Python` programming language. It is backward compatible to version 2.4. The recommended version of `Python` is 2.7.

The program code consists of 3 source files:

`temanejo.py`	main program, main window, communicator class, dialogues
`temanejo_graph.py`	`Graph` data structure
`temanejo_graphWidget.py`	widget for graph display, graph drawing routines ("view")

`temanejo.py` is the executable while the other three files are imported as modules.

5.2 Setting up the socket connection

TEMANEJO connects as a client socket to a server socket by the following commands:

temanejo.py: TemanejoWindow.on_connect()

```
self.sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
self.sock.connect((ayu_host, ayu_port))
```

It receives the host and port information from the user input by the initial connect dialogue or sets default parameters.

The server side socket has to be set up in advance. In C that could look like this:

C example: set up server socket

```
#include <sys/socket.h>
#include <netinet/in.h>
int server_socket = -1 , client_socket = -1;
struct sockaddr_in srv , cli;
int cli_size = sizeof(cli);
srv.sin_addr.s_addr = INADDR_ANY;
srv.sin_port = 5977;
srv.sin_family = AF_INET;
server_socket = socket( AF_INET, SOCK_STREAM, 0 );
bind( server_socket, (struct sockaddr*) &srv, sizeof(srv) );
listen( server_socket, 1 );
client_socket = accept( server_socket,
                        (struct sockaddr*) &cli,
                        (socklen_t*) &cli_size );
```

In the following we will describe the information passed from the runtime to TEMANEJO . We refer to this as the *information stream*

5.3 The information stream

In order for TEMANEJO to display the dependency graph it needs certain information from the application which it expects to receive as messages of a well defined binary format from the socket server. This protocol is described in the following.

5.3.1 The format

TEMANEJO sets up a thread which will continuously receive messages from the socket server. The expected format consists of 8 64-Bit-integers in big-endian (=network) byte order. In C one can compile these packages like this. Details like the location of the header depend on the operating system:

C example: pack event information

```c
#include <endian.h>
int64_t buf[8];
buf[0] = htobe64(runtimeId);
buf[1] = htobe64(taskId);
buf[2] = htobe64(event);
buf[3] = htobe64(0);
buf[4] = htobe64(0);
buf[5] = htobe64(0);
buf[6] = htobe64(0);
buf[7] = htobe64(ts);
```

The individual integers have distinct meanings explained in table 1. The only exception to this format occurs when the function name is passed after a `AYU_REGISTERTASK` event. Please refer to the events section (5.3.2) for details.

The **first three** and the **last** integer have a fixed meaning for each event message:

The **first number** is a identifier for the runtime environment. As of the present version of TEMANEJO , this number is not processed. It is reserved for distinguishing connections to several runtime environments.

The **second number** is the task id of the task which the message refers to. It is ignored when there is no specific task involved, such as initialisation and finalising the runtime environment.

The **third number** is the event id. The id for each event is given in section 5.3.2. It is of great use to define an enum (in C) as shown below:

C example: event id enum

```c
enum ayu_event_t {
  AYU_EVENT_NULL = 0,
  AYU_PREINIT = 1,
  AYU_INIT = 2,
  AYU_FINISH = 3,
  AYU_REGISTERFUNCTION = 4,
  AYU_ADDTASK = 5,
  AYU_ADDHIDDENTASK = 6,
```

```
  AYU_ADDDEPENDENCY = 7,
  AYU_ADDTASKTOQUEUE = 8,
  AYU_PRESELECTTASK = 9,
  AYU_PRERUNTASK = 10,
  AYU_RUNTASK = 11,
  AYU_POSTRUNTASK = 12,
  AYU_RUNTASKFAILED = 13,
  AYU_REMOVETASK = 14,
  AYU_WAITON = 15,
  AYU_BARRIER = 16,
  AYU_ADDWAITONTASK = 17
};
```

This code is taken from `Ayudame` which is a library used for communicating with the SMPSs runtime environment. Hence the prefix. Some of these events are not (yet/anymore) used in TEMANEJO . The relevant events are described below.

5.3.2 The events

`AYU_EVENT_NULL`: dummy event
optional for initialisation of event variables, event Id: ***0***

Does not represent an event.

`AYU_PREINIT`: Pre-initialisation
mandatory for logging, event Id: ***1***

Should be sent to TEMANEJO just after socket initialisation is done.

The additional data must be a pointer to an integer denoting which runtime is in use: CppSs (1), SMPSs (2), OMPSs (3), GOMP (4) or (CILK).

`AYU_INIT`: Initialisation
optional for logging, event Id: ***2***

Should be sent just after all initialisation is finished. This is a good moment to pause the execution and force the user of TEMANEJO to hit the "run" button in order to proceed to execute the parallel part of the application.

`AYU_FINISH`: Finalisation
recommended for closing socket connection, event Id: ***3***

This event notifies TEMANEJO that the parallel part of the application has finished. The socket connection will be closed.

`AYU_REGISTERFUNCTION`: Register taskified functions
optional for function name display, event Id: ***4***

position	meaning
0	runtime Id of the emitting runtime. This information is for future use.
1	task Id of the emitting task. If the master thread (i.e. no task) is emitting the event, task Id is 0.
2	event Id. For a description of the events see below.
3	in case of ▷ `AYU_REGISTERTASK`: length of the function name. The function name will be given in the subsequent message. ▷ `AYU_ADDTASK`: function Id, -1 for hidden tasks. ▷ `AYU_ADDDEPENDENCY`: task Id of the predecessor. ▷ `AYU_PRERUNTASK` and `AYU_ADDTASKTOQUEUE`: thread Id of the thread which will execute the task or in whose queue the task is queued. ▷ 0 otherwise.
4	in case of ▷ `AYU_REGISTERTASK`: function Id. ▷ `AYU_ADDDEPENDENCY`: address of the dependency, sets the colour of the graph edges. ▷ 0 otherwise.
5	in case of ▷ `AYU_ADDDEPENDENCY`: original address of the dependency. ▷ 0 otherwise.
6	reserved for future use.
7	timestamp.

Table 1: Description of the data format for the information stream.

This event consists of two messages. In the first one, the event is passed along with the length of the function name and the function Id, in the second the function name is passed as a `\0` terminated string. Hence to register the function `foo` with function Id 23 at time 43 these messages have to be sent:

$$[0, 4, 3, 23, 0, 0, 0, 43], ['\text{foo}']$$

It has to be made sure (e.g. by a mutex lock) that the 2nd message is sent directly after the first.

`AYU_ADDTASK`: Create task

mandatory for graph display, event Id: ***5***

A task is represented by a node in the dependency graph in TEMANEJO . When a task is created a notification should be passed to TEMANEJO containing the task Id and the function Id of the function executed by the task. If the function Id is unknown to TEMANEJO a new function reference is created with the name "function <Id>". If a function with the same Id is registered later, the counters will respond in an arbitrary fashion. Therefore it is recommended to either make sure that every function is registered *before* the first task for this function is created, or *not* to register function names at all.

The task is created with the status "not queued", which is represented in TEMANEJO as a red margin around the task node by default.

The additional data must be a pointer to an integer denoting whether the task is critical (high-priority, =1) or not (=0).

`AYU_ADDHIDDENTASK`: Add hidden task

do not use, development in progress, event Id: ***6***

Notify TEMANEJO of a task which does not represent a taskified function. May be used for wait-on events.
Do not use! This is under development!

`AYU_ADDDEPENDENCY`: Define dependency

mandatory for graph display, event Id: ***7***

A dependency is represented by an edge in the dependency graph. When a dependency is created, the application should notify TEMANEJO passing along the task Id of the predecessor, the memory address of the variable or array causing the dependency and the original memory address of the variable (in case of renaming). If the address is 0 a black edge without label will be drawn. This is recommended if no address information is to be displayed.

It is necessary that both tasks (the dependant and the task it depends on) are already created when defining a dependency.

`AYU_ADDTASKTOQUEUE`: Queue task notification

recommended for graph display, event Id: ***8***

When a task does not depend on any other tasks it can be marked as queued. It will be drawn with a yellow margin around the node in TEMANEJO by default. The task must exist prior to this event. Any number of tasks can be queued at the same time. Additionally the Id of the thread which queued the task can be passed.

`AYU_PRESELECTTASK`**: Task is about to be dequeued**
do not use, debugging event, event Id: ***9***

This event notifies TEMANEJO that the runtime is checking the queue(s) for a task to run. As this event will usually occur many times, it is completely ignored, i.e. not even logged.

`AYU_PRERUNTASK`**: Prerun task notification**
recommended for graph display, event Id: ***10***

When a task is scheduled for execution, i.e. an execution thread has dequeued the task it can be marked as "about to run". This event is recommended to be emitted before pausing the execution when stepping through the dependency graph. By default TEMANEJO will draw a green margin around the node. The number of nodes marked this way is at any time equal or less the number of execution threads. Additionally the Id of the thread which runs the task can be passed.

`AYU_RUNTASK`**: Run task notification**
optional for time measurement, event Id: ***11***

Just before the task is executed a "run task" event can be emitted. This event is intended for time measurement. Between this event and the actual execution of the task there should be as less code as possible to ensure exact time stamps.

`AYU_POSTRUNTASK`**: Task execution finished**
optional for logging, event Id: ***12***

Ignored by TEMANEJO .

`AYU_RUNTASKFAILED`**: Task execution failed**
optional for failed tasks, event Id: ***13***

Ignored by TEMANEJO .

`AYU_REMOVETASK`**: Task execution finished**
recommended for graph display and time measurement, event Id: ***14***

When a task has finished this event should be passed to TEMANEJO . The task will be drawn without a margin by default. When "task duration" is displayed, it is calculated as difference between the time stamp of this event and the "run task" event.

`AYU_WAITON`: **Wait-on event**
do not use, development in progress, event Id: ***15***

For treating wait-on events.
Do not use! This is under development!

`AYU_BARRIER`: **Barrier notification**
recommended for graph display with horizontal barrier lines, event Id: ***16***

Notifies TEMANEJO of a barrier. Barriers are drawn as horizontal lines. The following tasks will start a new graph below the barrier line.

`AYU_ADDWAITONTASK`: **Wait-on event**
do not use, development in progress, event Id: ***17***

For treating wait-on events.
Do not use! This is under development!